

Transfer and Multi-Task Learning

CS 285

Instructor: Sergey Levine
UC Berkeley



What's the problem?

this is easy (mostly)



this is impossible



Why?

Montezuma's revenge



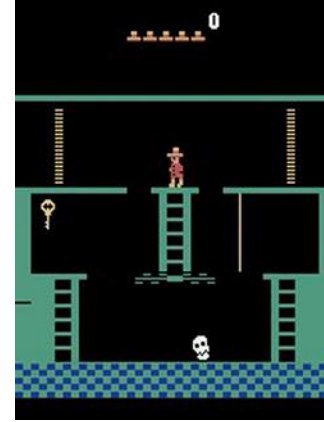
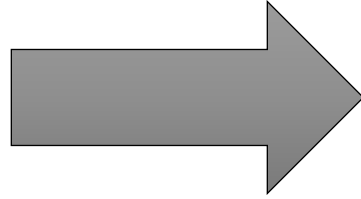
- Getting key = reward
- Opening door = reward
- Getting killed by skull = bad

Montezuma's revenge



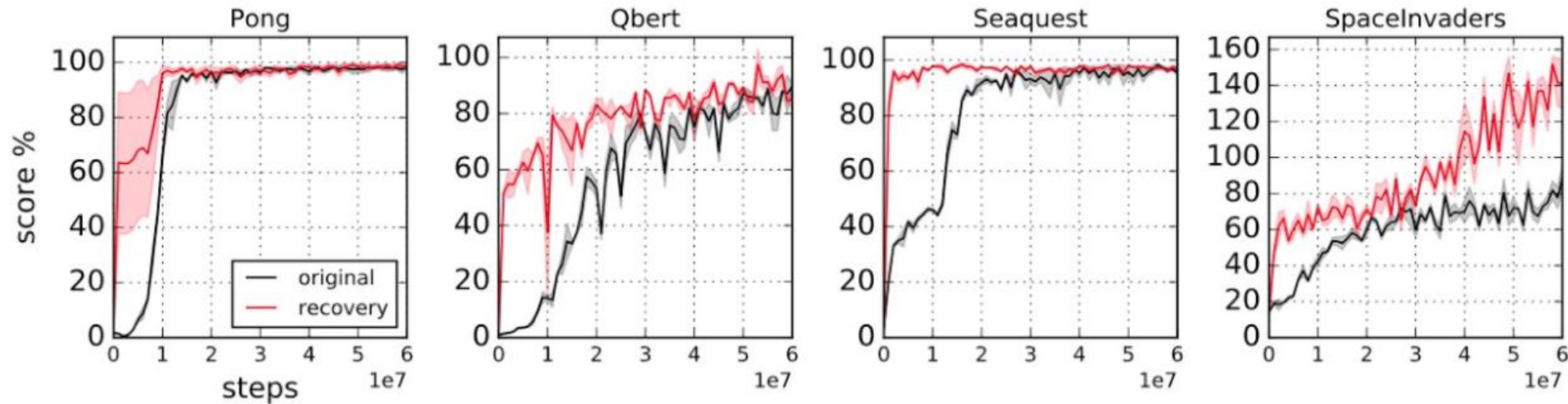
- We know what to do because we **understand** what these sprites mean!
- Key: we know it opens doors!
- Ladders: we know we can climb them!
- Skull: we don't know what it does, but we know it can't be good!
- **Prior understanding of problem structure can help us solve complex tasks quickly!**

Can RL use the same prior knowledge as us?



- If we've solved prior tasks, we might acquire useful knowledge for solving a new task
- How is the knowledge stored?
 - Q-function: tells us which actions or states are good
 - Policy: tells us which actions are potentially useful
 - some actions are never useful!
 - Models: what are the laws of physics that govern the world?
 - Features/hidden states: provide us with a good representation
 - Don't underestimate this!

Aside: the representation bottleneck



To decouple reinforcement learning from representation learning, we decapitate an agent by destroying its policy and value outputs and then re-train end-to-end.

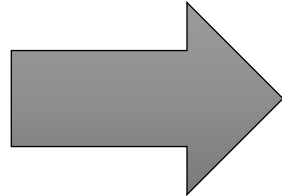
The representation remains and the policy is swiftly recovered. **The gap between initial optimization and recovery shows a representation learning bottleneck.**

Transfer learning terminology

transfer learning: using experience from one set of tasks for faster learning and better performance on a new task

in RL, **task** = **MDP**!

source domain



target domain



“shot”: number of attempts in the target domain

0-shot: just run a policy trained in the source domain

1-shot: try the task once

few shot: try the task a few times

How can we frame transfer learning problems?

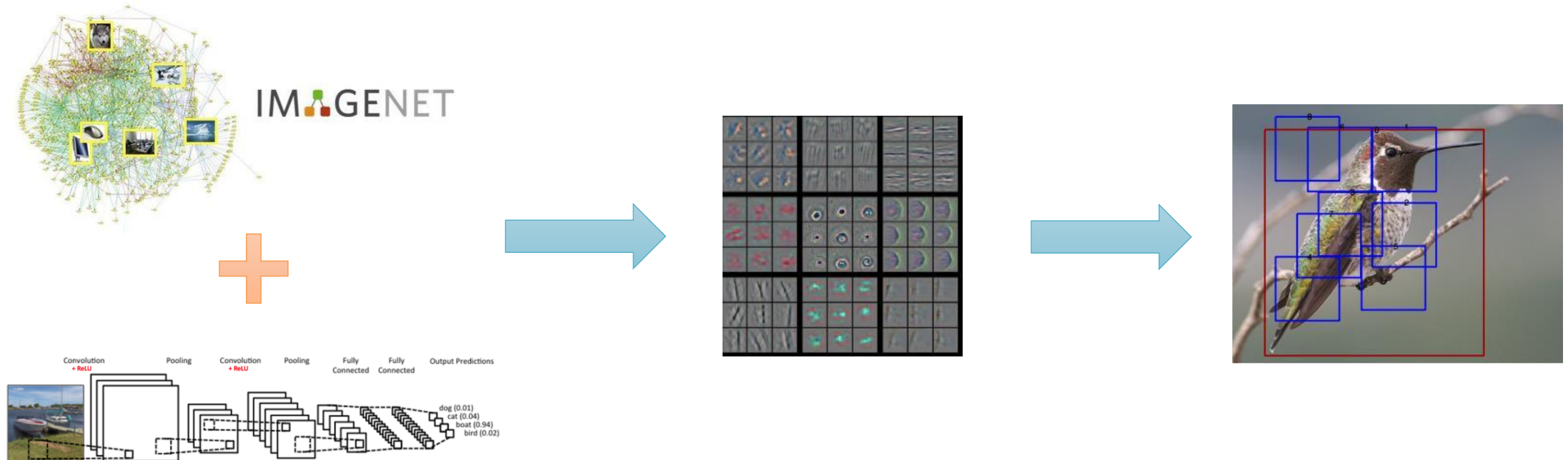
No single solution! Survey of various recent research papers

1. Forward transfer: train on one task, transfer to a new task
 - a) Transferring visual representations & domain adaptation
 - b) Domain adaptation in reinforcement learning
 - c) Randomization
2. Multi-task transfer: train on many tasks, transfer to a new task
 - a) Sharing representations and layers across tasks in multi-task learning
 - b) Contextual policies
 - c) Optimization challenges for multi-task learning
 - d) Algorithms
3. Transferring models and value functions
 - a) Model-based RL as a mechanism for transfer
 - b) Successor features & representations

Forward Transfer

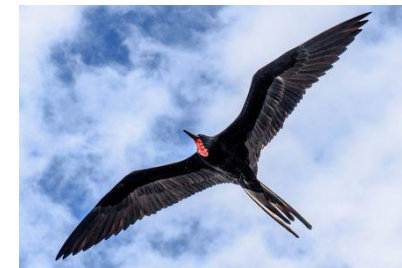
Pretraining + Finetuning

The most popular transfer learning method in (supervised) deep learning!

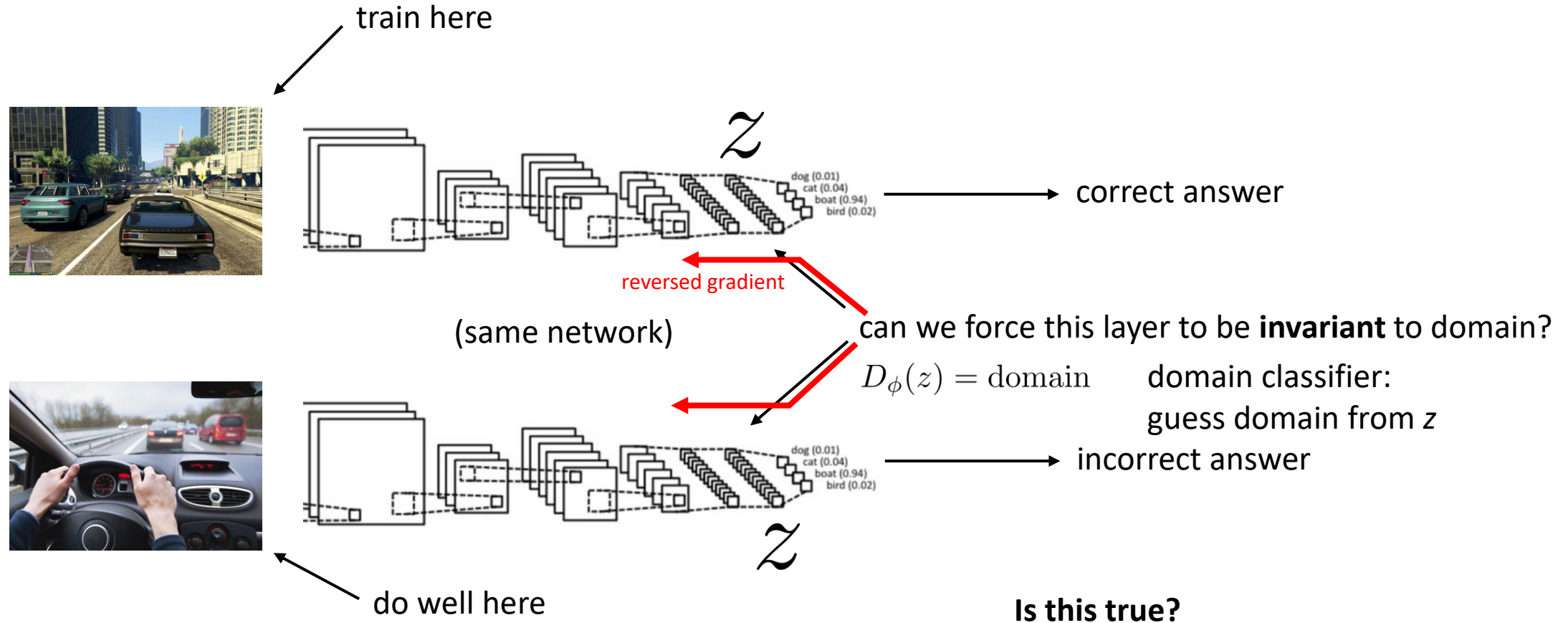


What issues are we likely to face?

- **Domain shift:** representations learned in the source domain might not work well in the target domain
- **Difference in the MDP:** some things that are possible to do in the source domain are not possible to do in the target domain
- **Finetuning issues:** if pretraining & finetuning, the finetuning process may still need to explore, but optimal policy during finetuning may be deterministic!



Domain adaptation in computer vision

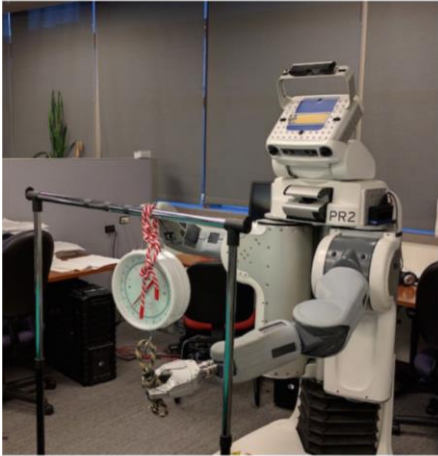


Invariance assumption: everything that is **different** between domains is **irrelevant**

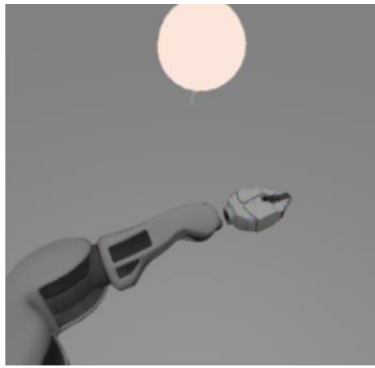
formally:

$p(x)$ is different exists some $z = f(x)$ such that $p(y|z) = p(y|x)$, but $p(z)$ is same

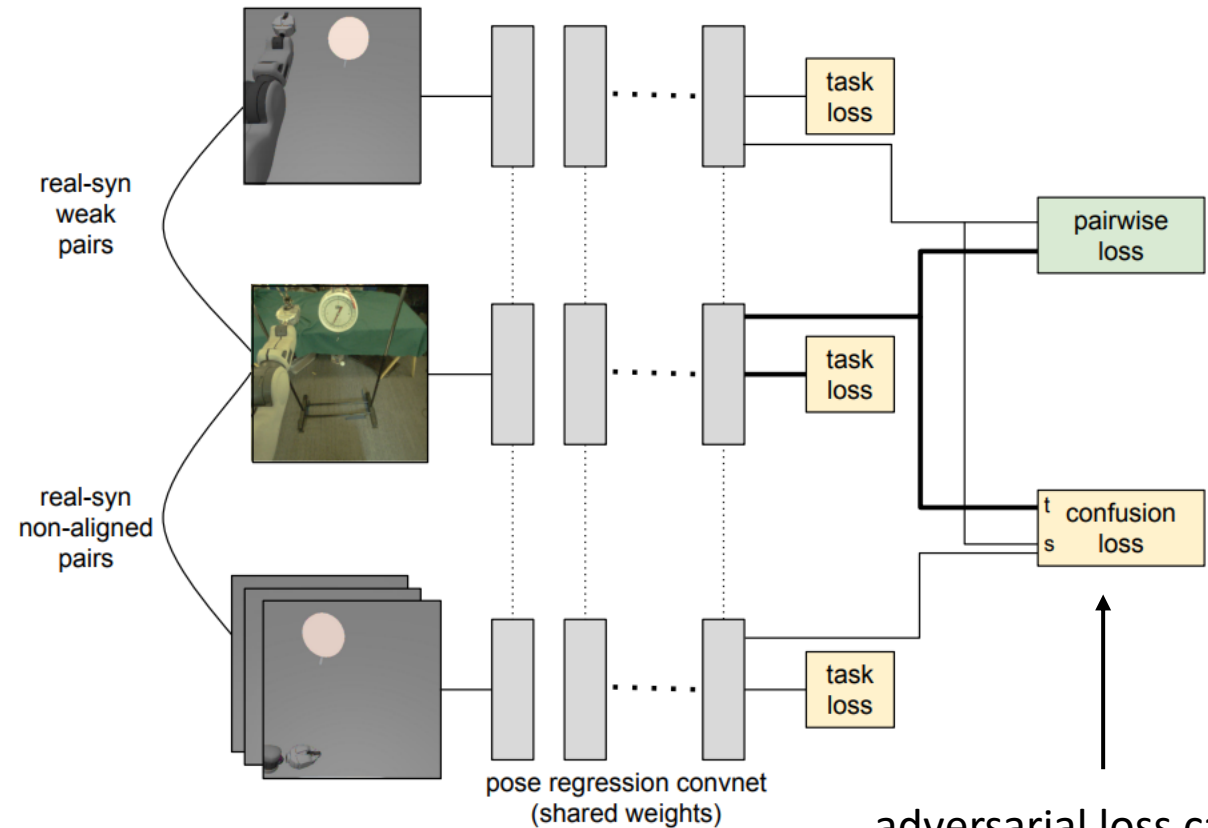
How do we apply this idea in RL?



simulated images



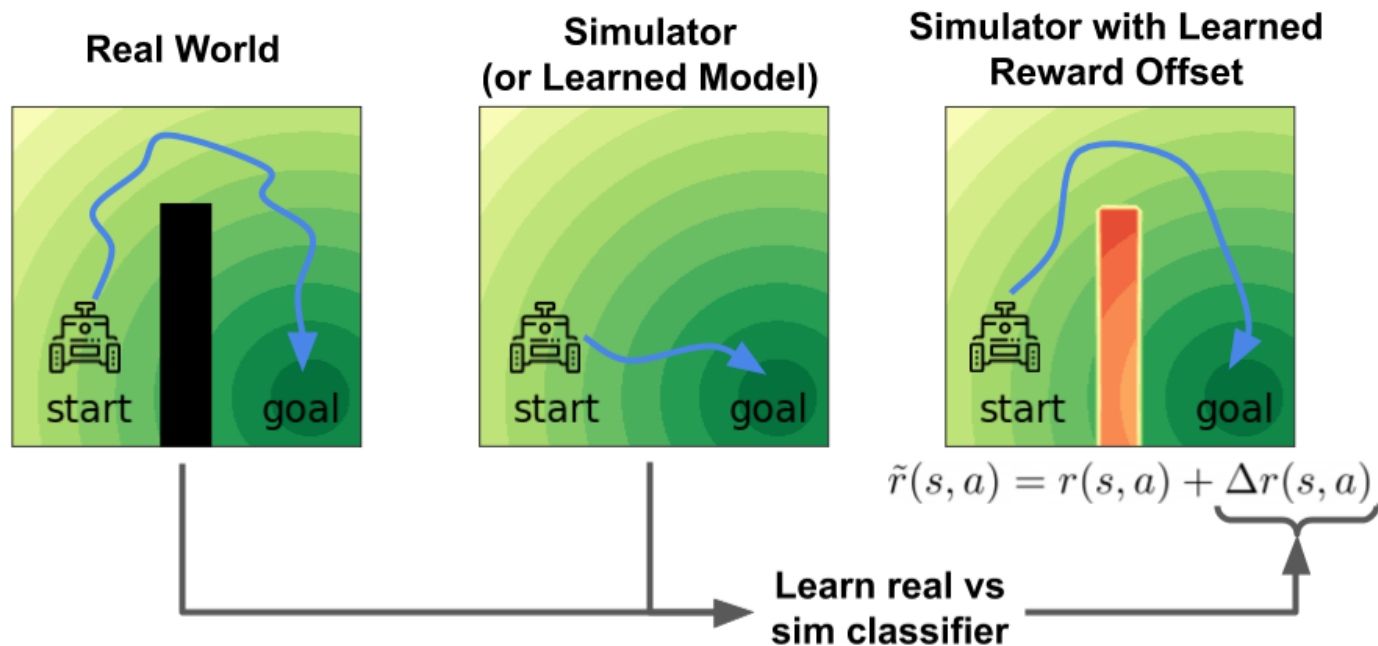
real images



adversarial loss causes
internal CNN features to be
indistinguishable for sim and real

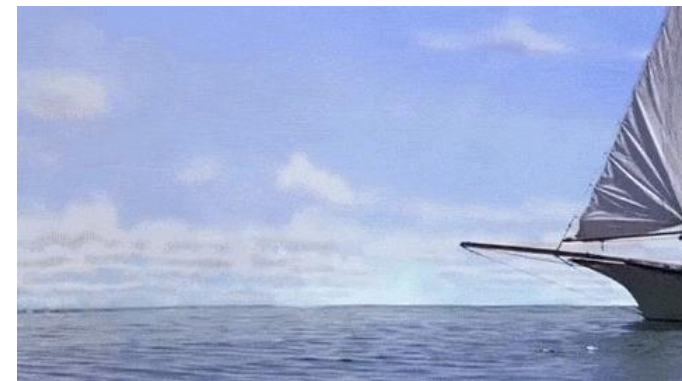
Domain adaptation in RL for dynamics?

Why is **invariance** not enough when the dynamics don't match?



$$\Delta r(s_t, a_t, s_{t+1}) = \log p_{\text{target}}(s_{t+1} | s_t, a_t) - \log p_{\text{source}}(s_{t+1} | s_t, a_t).$$

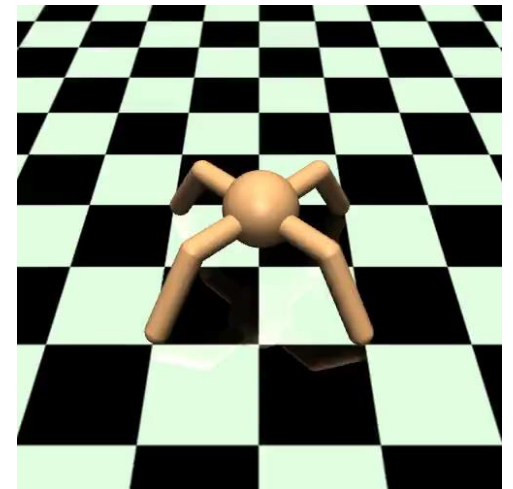
$$\begin{aligned} \Delta r(s_t, a_t, s_{t+1}) = & \log p(\text{target} | s_t, a_t, s_{t+1}) - \log p(\text{target} | s_t, a_t) \\ & - \log p(\text{source} | s_t, a_t, s_{t+1}) + \log p(\text{source} | s_t, a_t) \end{aligned}$$



When might this **not** work?

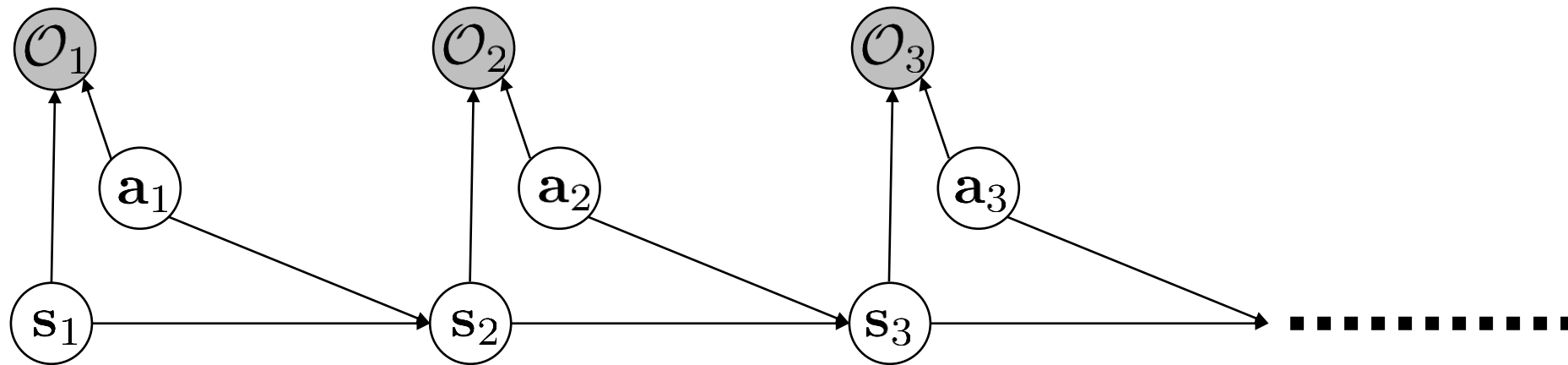
What if we can also finetune?

1. RL tasks are generally much less diverse
 - Features are less general
 - Policies & value functions become overly specialized
2. Optimal policies in fully observed MDPs are deterministic
 - Loss of exploration at convergence
 - Low-entropy policies adapt very slowly to new settings



Finetuning with maximum-entropy policies

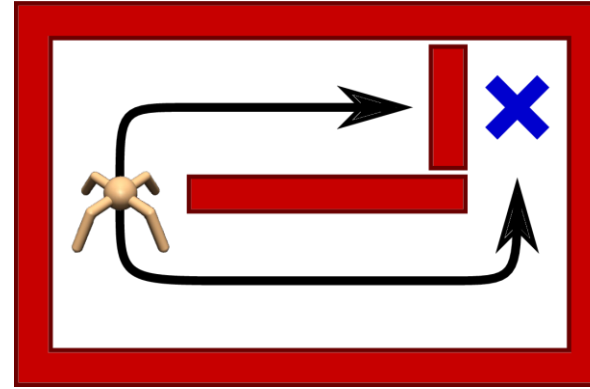
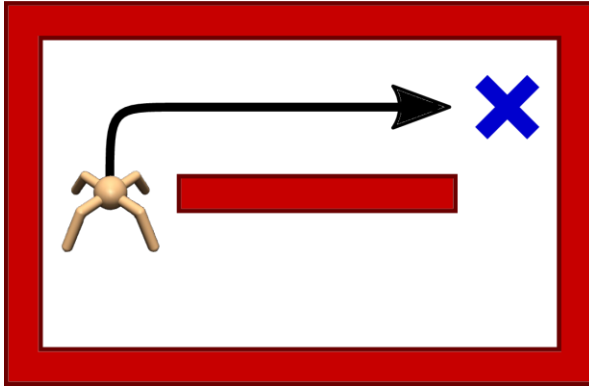
How can we increase diversity and entropy?



$$\pi(\mathbf{a}|\mathbf{s}) = \exp(Q_\phi(\mathbf{s}, \mathbf{a}) - V(\mathbf{s})) \text{ optimizes } \sum_t E_{\pi(\mathbf{s}_t, \mathbf{a}_t)}[r(\mathbf{s}_t, \mathbf{a}_t)] + \underbrace{E_{\pi(\mathbf{s}_t)}[\mathcal{H}(\pi(\mathbf{a}_t|\mathbf{s}_t))]}_{\text{policy entropy}}$$

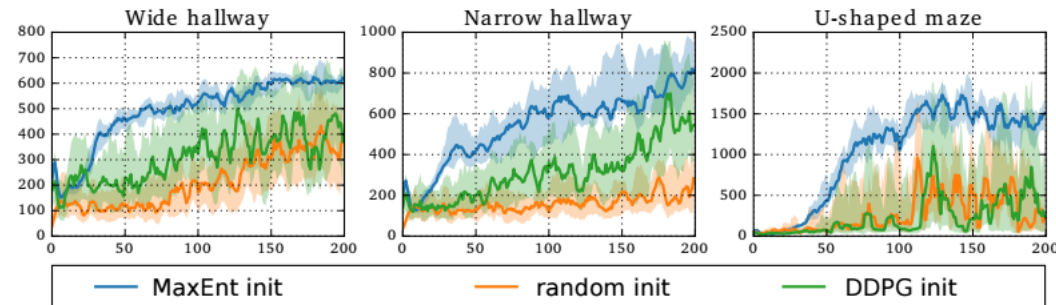
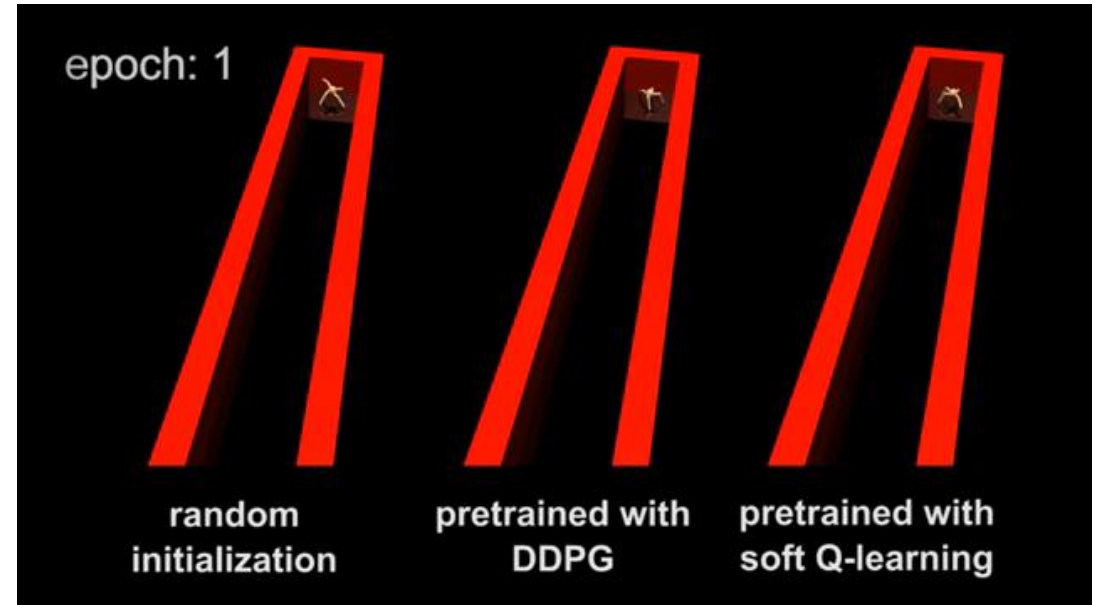
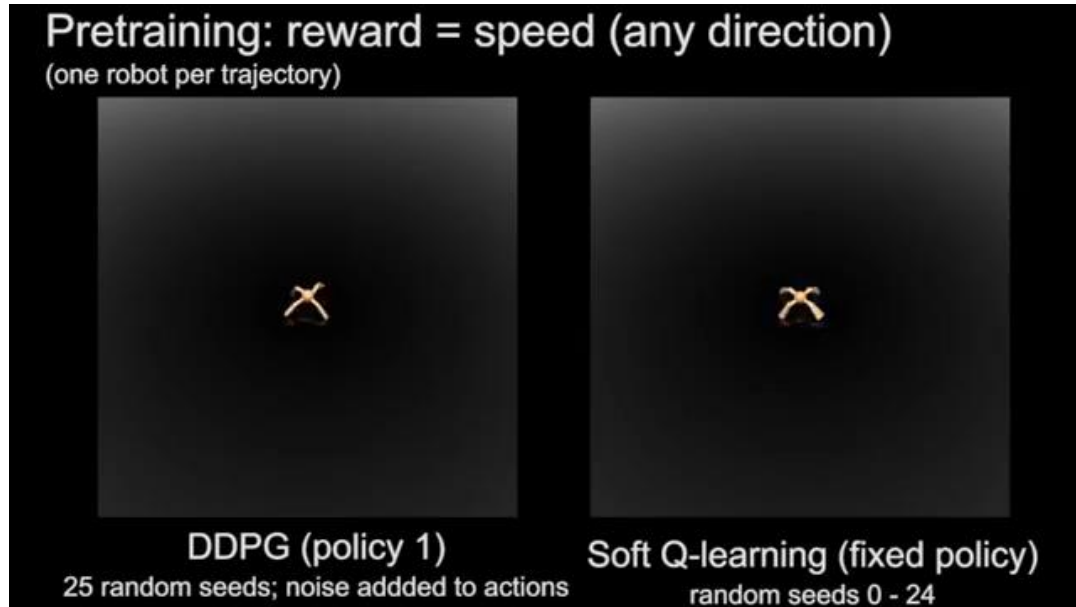
Act as randomly as possible while collecting high rewards!

Example: pre-training for robustness



Learning to **solve a task in all possible ways** provides for more robust transfer!

Example: pre-training for diversity



Domain adaptation: suggested readings

Tzeng, Hoffman, Zhang, Saenko, Darrell. **Deep Domain Confusion: Maximizing for Domain Invariance**. 2014.

Ganin, Ustinova, Ajakan, Germain, Larochelle, Laviolette, Marchand, Lempitsky. **Domain-Adversarial Training of Neural Networks**. 2015.

Tzeng*, Devin*, et al., **Adapting Visuomotor Representations with Weak Pairwise Constraints**. 2016.

Eysenbach et al., **Off-Dynamics Reinforcement Learning: Training for Transfer with Domain Classifiers**. 2020.

...and many many others!

Finetuning: suggested readings

Finetuning via MaxEnt RL: Haarnoja*, Tang*, et al. (2017). **Reinforcement Learning with Deep Energy-Based Policies.**

Andreas et al. **Modular multitask reinforcement learning with policy sketches.** 2017.

Florensa et al. **Stochastic neural networks for hierarchical reinforcement learning.** 2017.

Kumar et al. **One Solution is Not All You Need: Few-Shot Extrapolation via Structured MaxEnt RL.** 2020

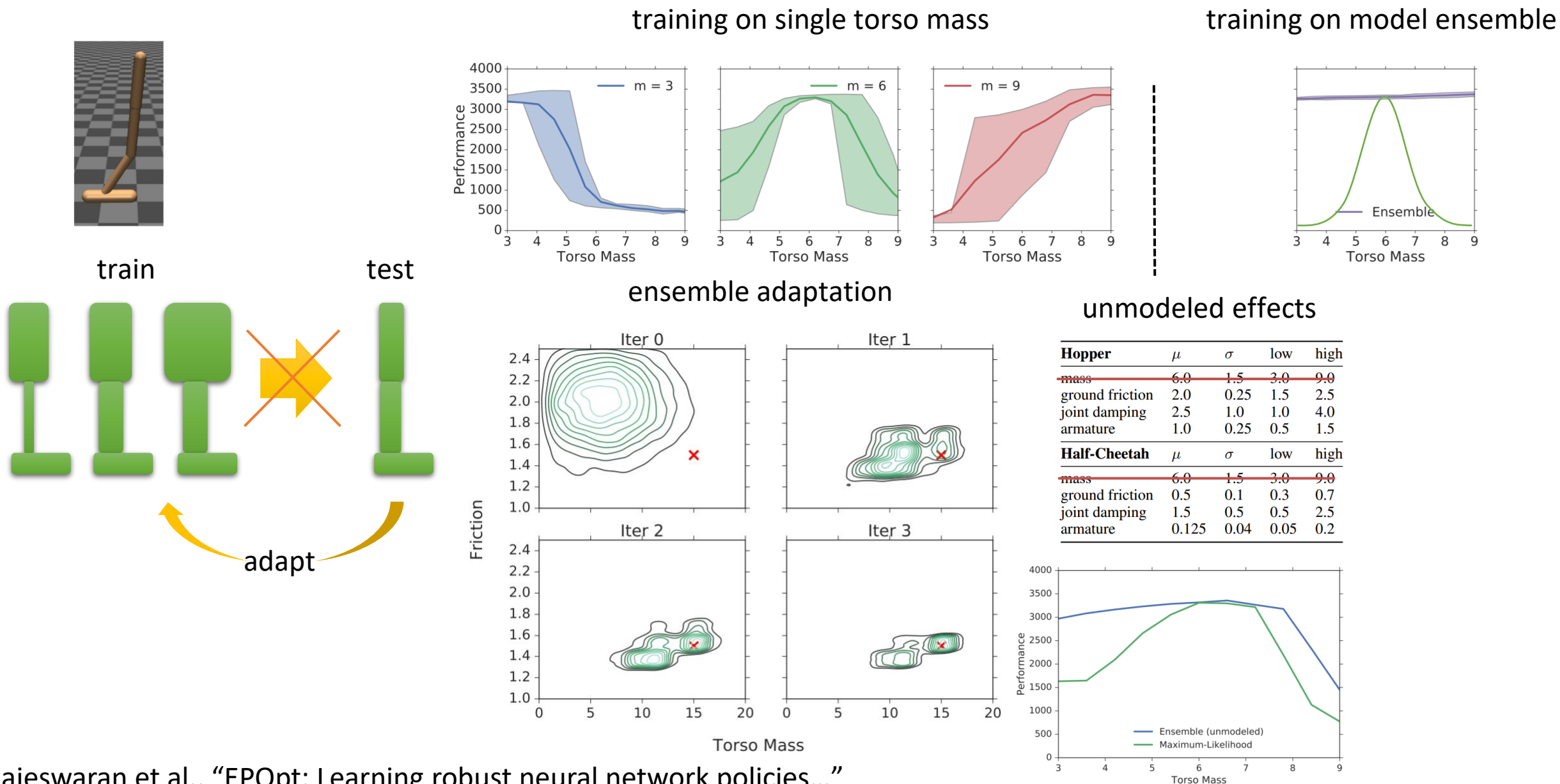
...and many many others!

Forward Transfer with Randomization

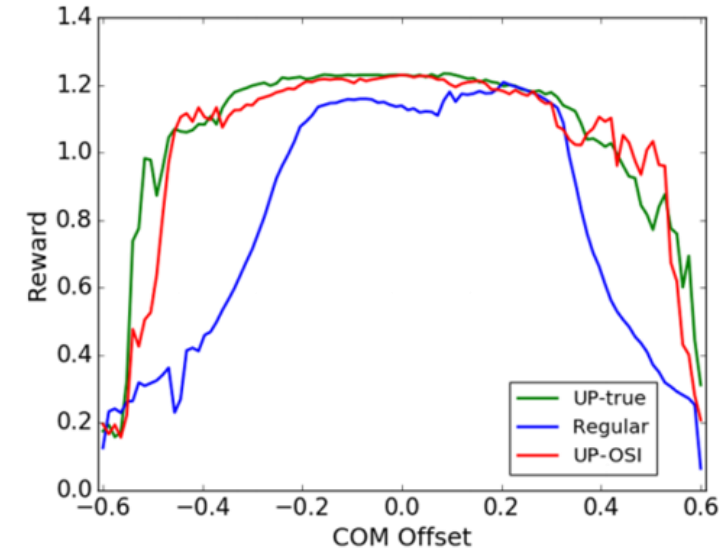
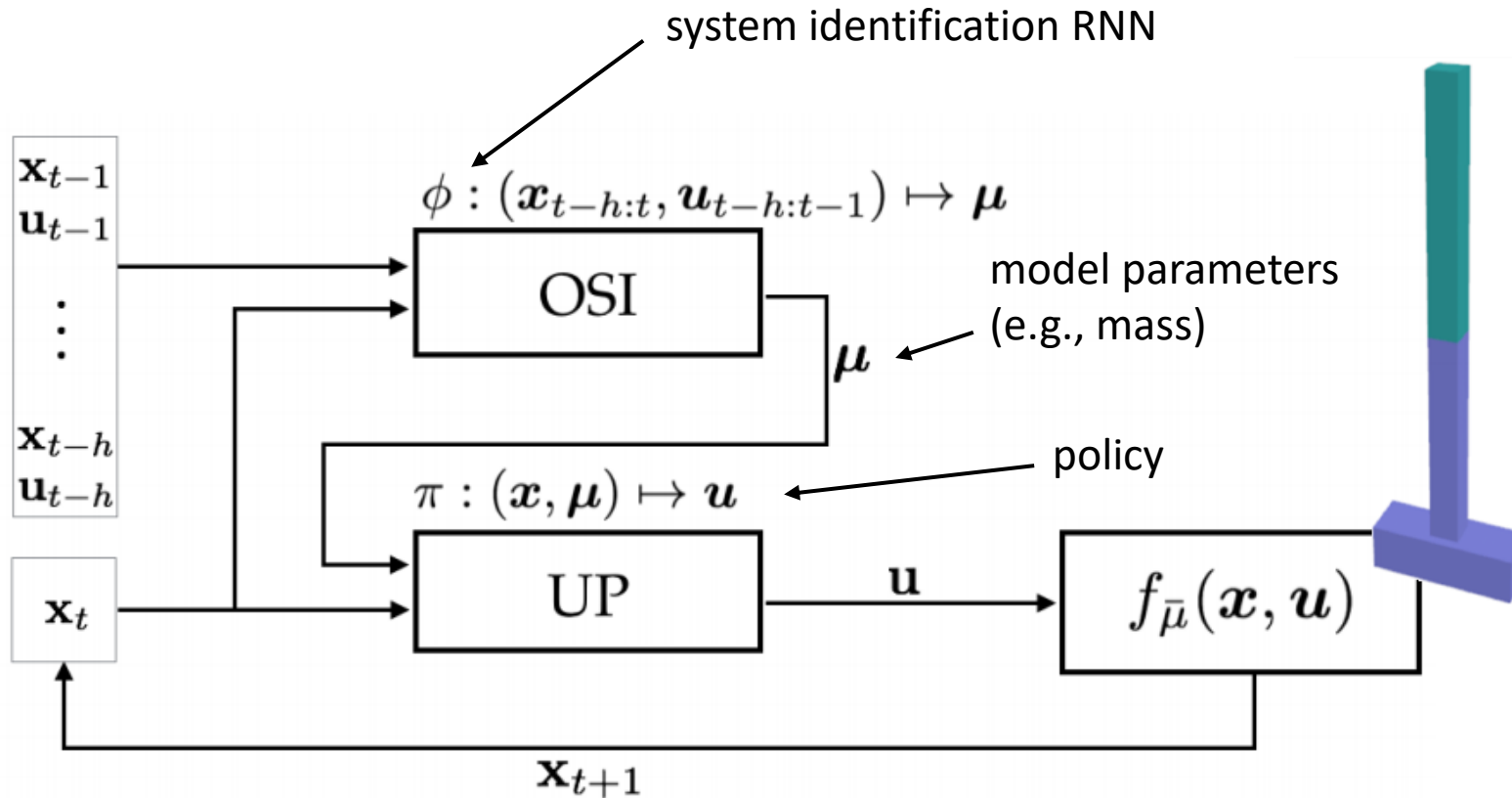
What if we can manipulate the source domain?

- So far: source domain (e.g., empty room) and target domain (e.g., corridor) are fixed
- What if we can **design** the source domain, and we have a **difficult** target domain?
 - Often the case for simulation to real world transfer

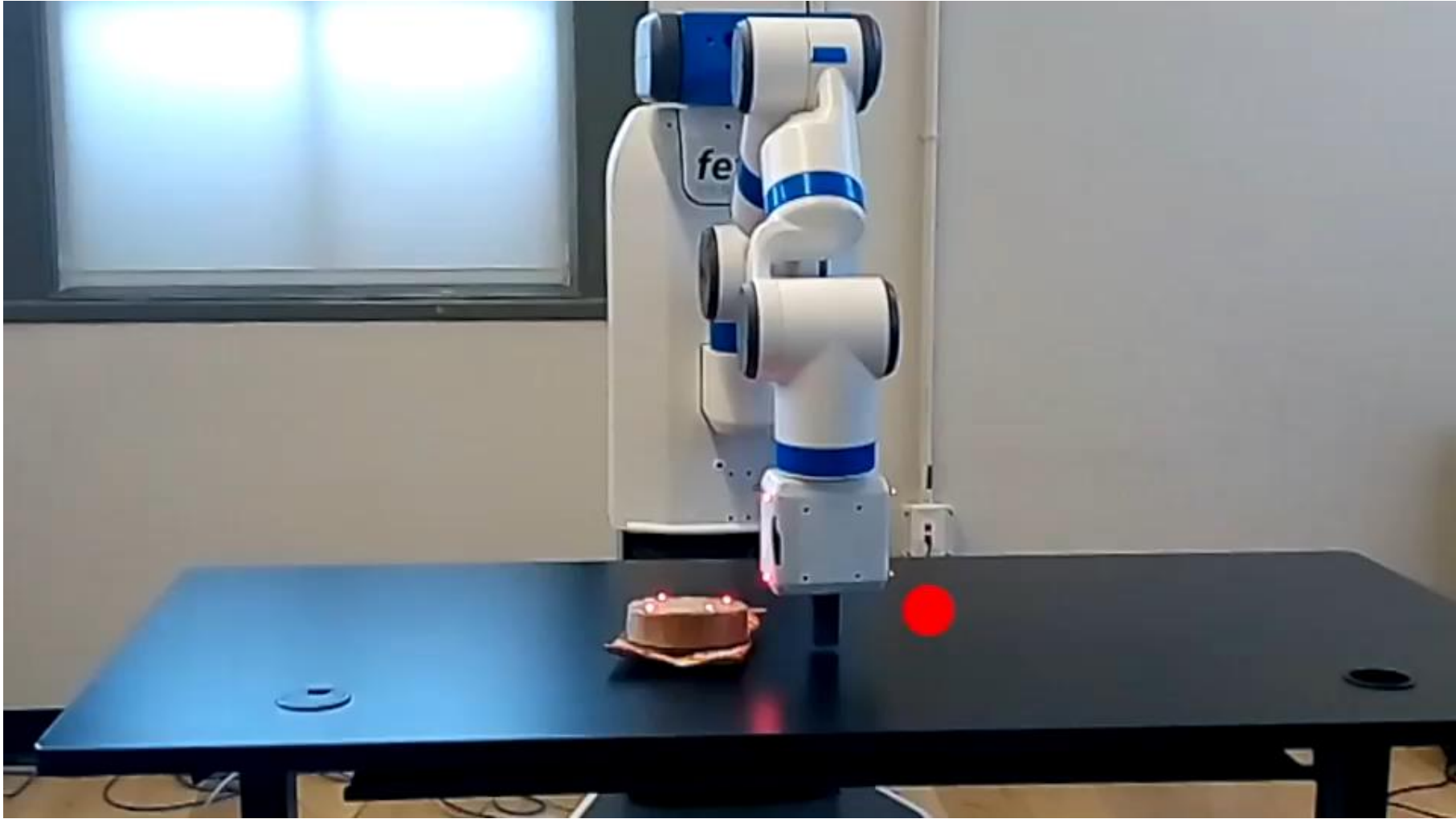
EPOpt: randomizing physical parameters



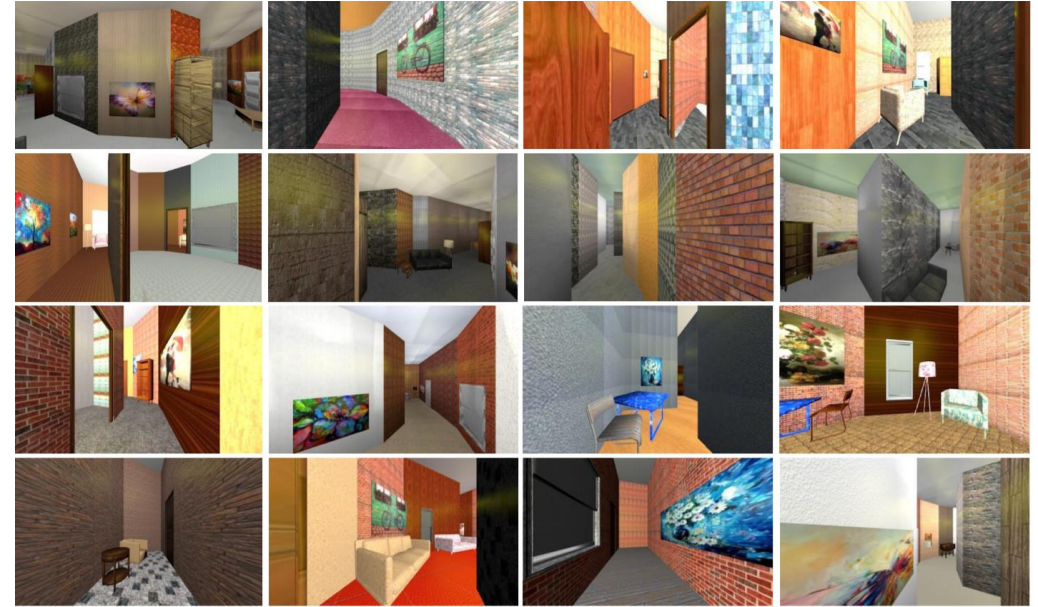
Preparing for the unknown: explicit system ID



Another example

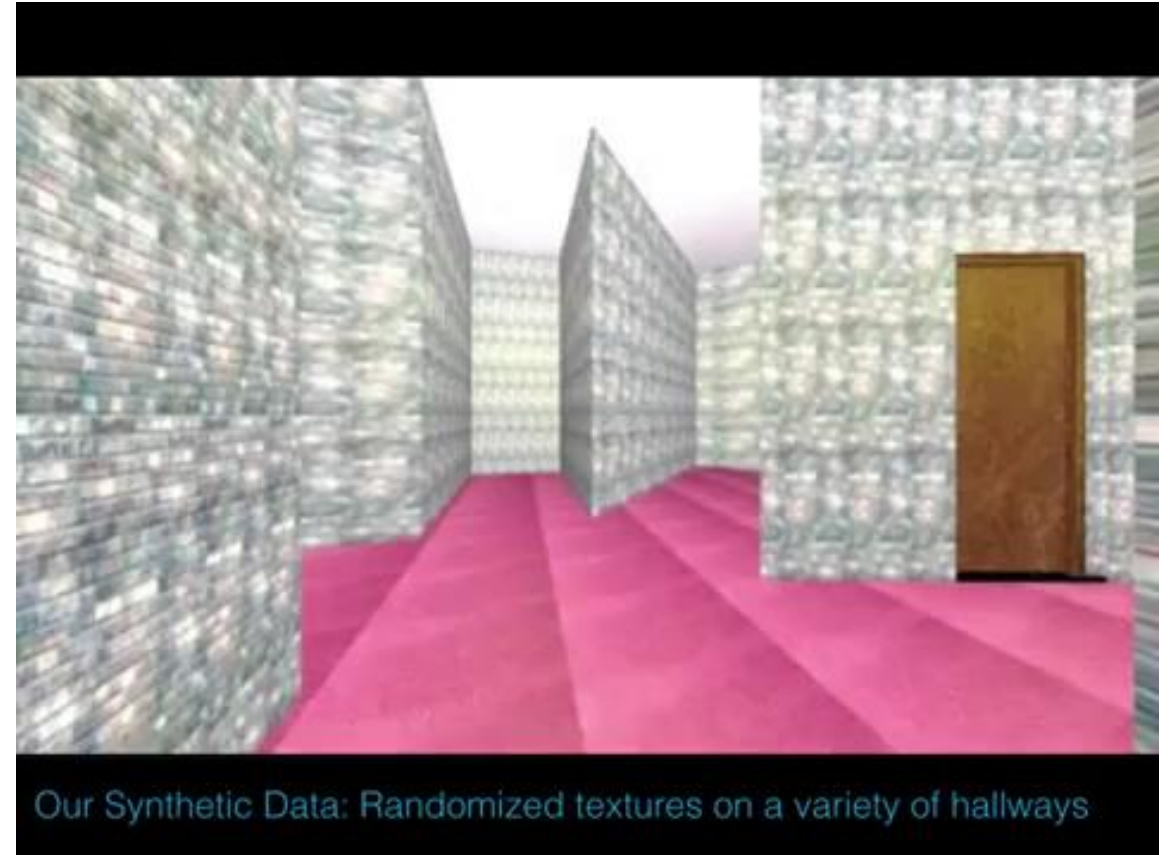


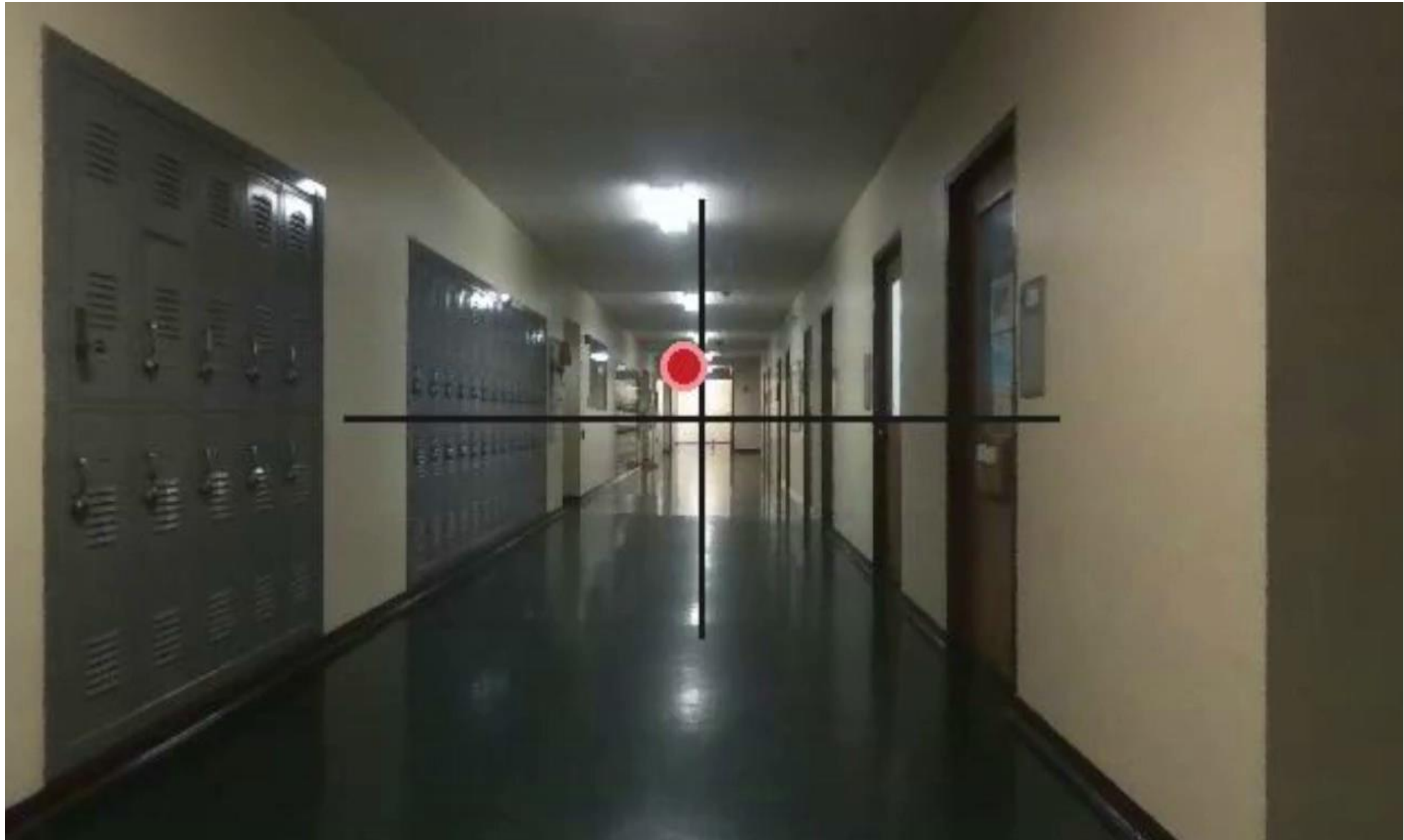
CAD2RL: randomization for real-world control



also called domain randomization

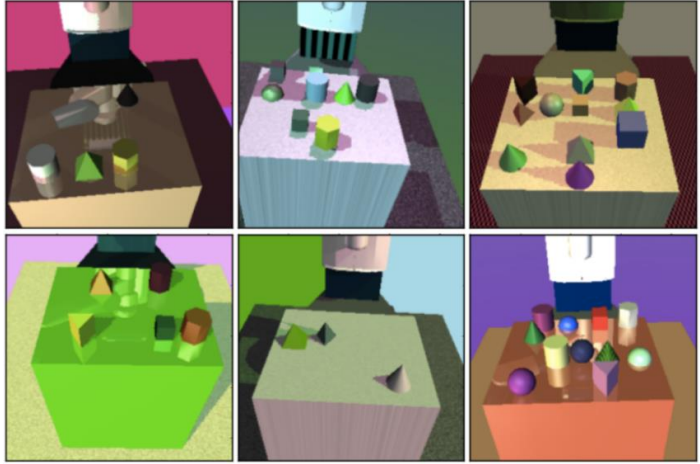
CAD2RL: randomization for real-world control



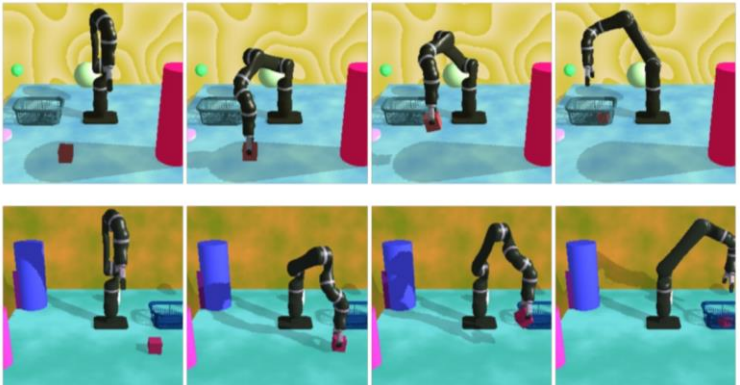


Sadeghi et al., "CAD2RL: Real Single-Image Flight without a Single Real Image"

Randomization for manipulation



Tobin, Fong, Ray, Schneider, Zaremba, Abbeel



James, Davison, Johns

Source domain randomization and domain adaptation suggested readings

Rajeswaran, et al. (2017). **EPOpt: Learning Robust Neural Network Policies Using Model Ensembles.**

Yu et al. (2017). **Preparing for the Unknown: Learning a Universal Policy with Online System Identification.**

Sadeghi & Levine. (2017). **CAD2RL: Real Single Image Flight without a Single Real Image.**

Tobin et al. (2017). **Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World.**

James et al. (2017). **Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task.**

Methods that **also** incorporate domain adaptation together with randomization:

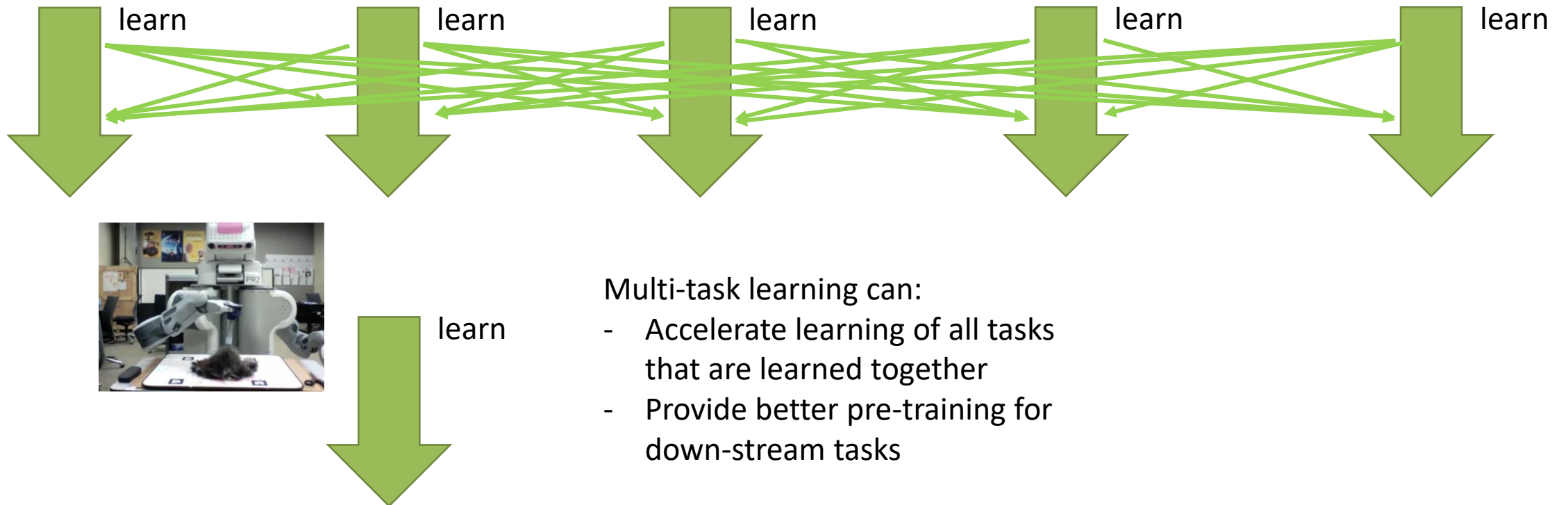
Bousmalis et al. (2017). **Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping.**

Rao et al. (2017). **RL-CycleGAN: Reinforcement Learning Aware Simulation-To-Real.**

... and many many others!

Multi-Task Transfer

Can we learn **faster** by learning multiple tasks?

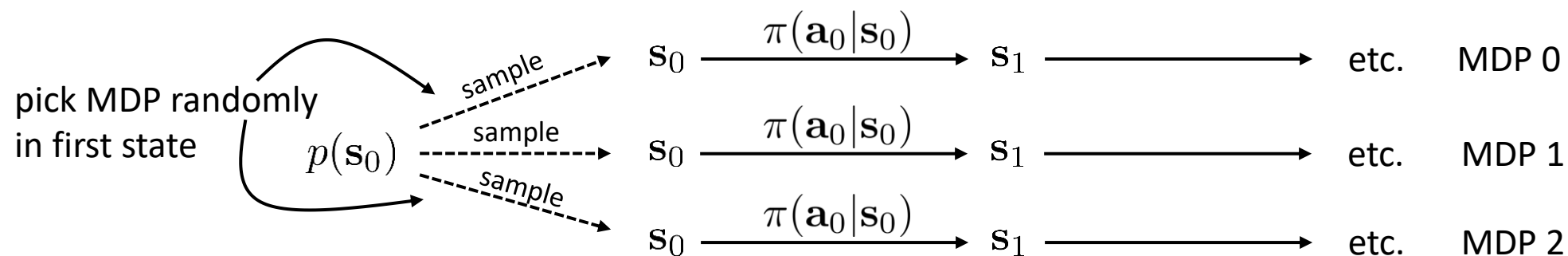


Multi-task learning can:

- Accelerate learning of all tasks that are learned together
- Provide better pre-training for down-stream tasks

Can we solve multiple tasks at once?

Multi-task RL corresponds to single-task RL in a **joint MDP**



What is difficult about this?

- **Gradient interference:** becoming better on one task can make you worse on another
 - **Winner-take-all problem:** imagine one task starts getting good – algorithm is likely to prioritize that task (to increase average expected reward) at the expense of others
- In practice, this kind of multi-task RL is very challenging

Actor-mimic and policy distillation

Goal: learn a single policy that can play all Atari games

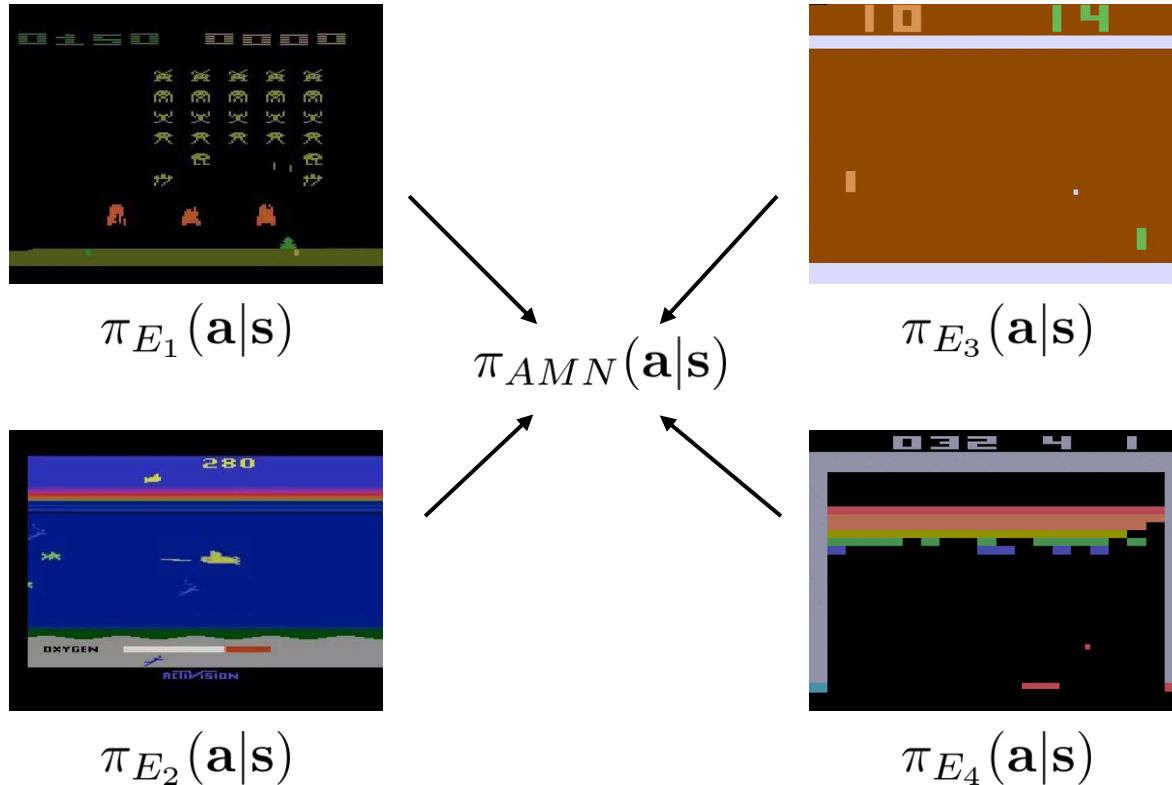
POLICY DISTILLATION

**Andrei A. Rusu, Sergio Gómez Colmenarejo, Çağlar Gülçehre*, Guillaume Desjardins,
James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu & Raia Hadsel**
Google DeepMind

ACTOR-MIMIC DEEP MULTITASK AND TRANSFER REINFORCEMENT LEARNING

Emilio Parisotto, Jimmy Ba, Ruslan Salakhutdinov
Department of Computer Science
University of Toronto

Distillation for Multi-Task Transfer



$$\mathcal{L} = \sum_{\mathbf{a}} \pi_{E_i}(\mathbf{a}|\mathbf{s}) \log \pi_{AMN}(\mathbf{a}|\mathbf{s})$$

(just supervised learning/distillation)

analogous to **guided policy search**, but
for transfer learning

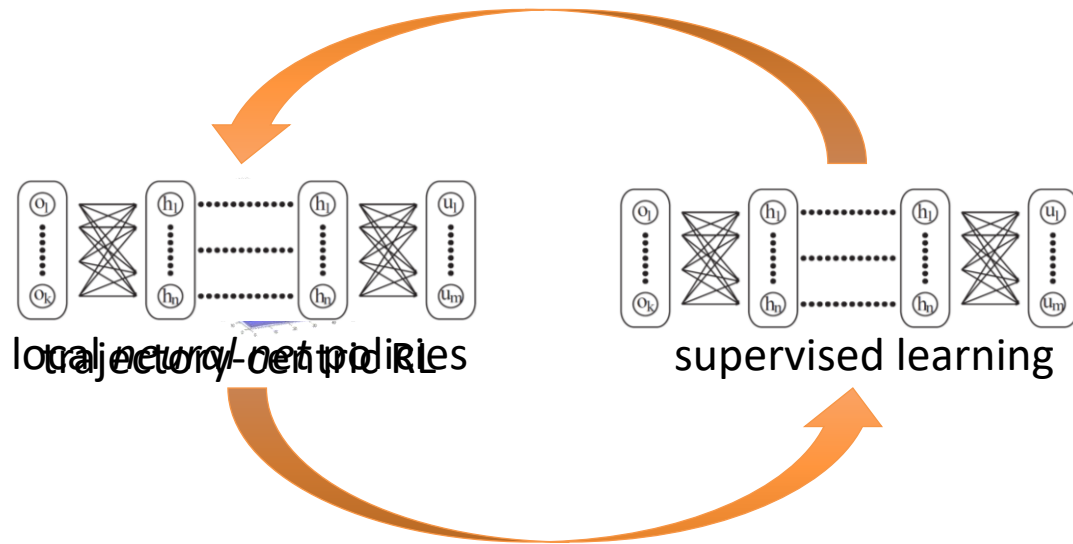
-> see model-based RL slides

some other details

(e.g., feature regression objective)

– see paper

Combining weak policies into a strong policy



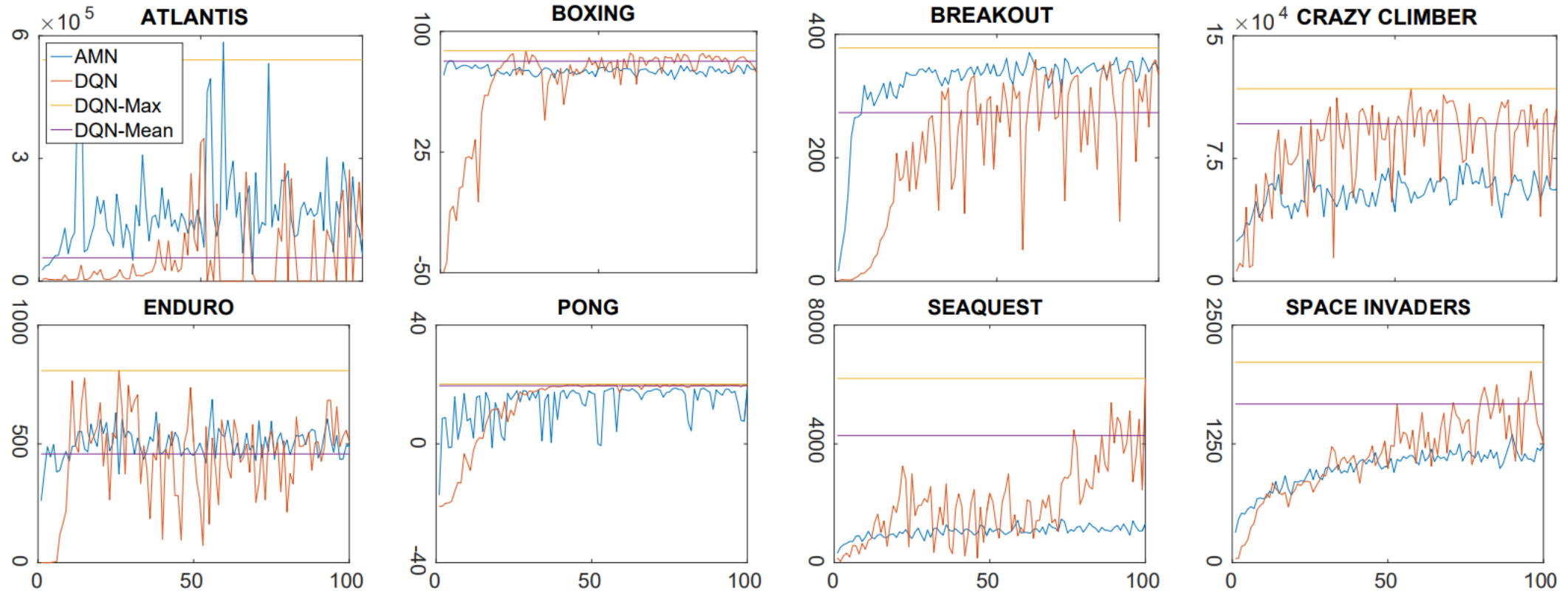
Divide and Conquer Reinforcement Learning

Divide and conquer reinforcement learning algorithm sketch:

1. optimize each local policy $\pi_{\theta_i}(\mathbf{a}_t|\mathbf{s}_t)$ on initial state $\mathbf{s}_{0,i}$ w.r.t. $\tilde{r}_{k,i}(\mathbf{s}_t, \mathbf{a}_t)$
2. use samples from step (1) to train $\pi_{\theta}(\mathbf{u}_t|\mathbf{x}_t)$ to mimic each $\pi_{\theta_i}(\mathbf{u}_t|\mathbf{x}_t)$
3. update reward function $\tilde{r}_{k+1,i}(\mathbf{x}_t, \mathbf{u}_t) = r(\mathbf{x}_t, \mathbf{u}_t) + \lambda_{k+1,i} \log \pi_{\theta}(\mathbf{u}_t|\mathbf{x}_t)$

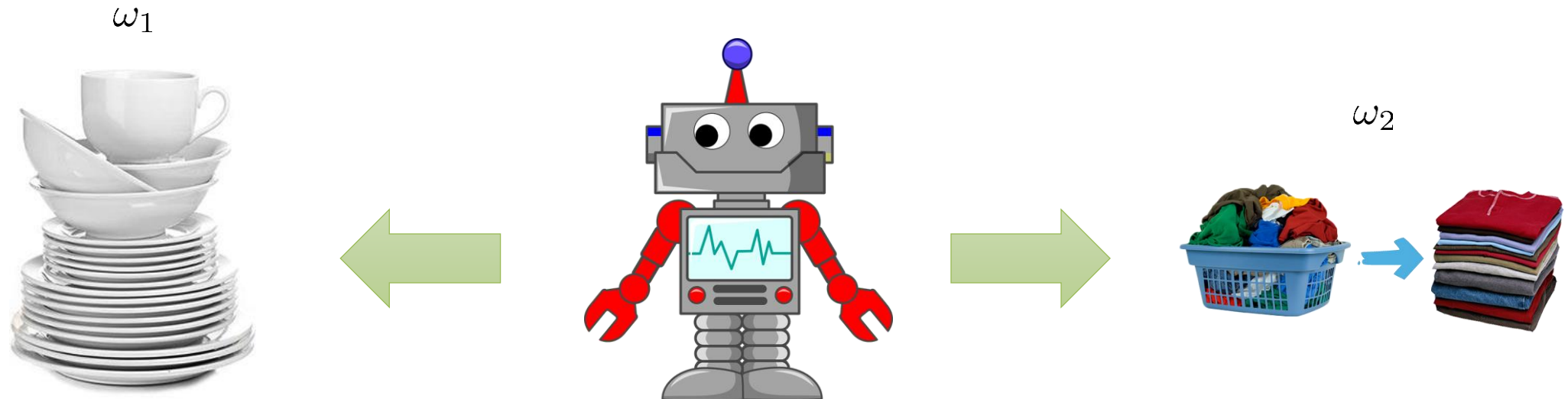
For details, see: “Divide and Conquer Reinforcement Learning”

Distillation Transfer Results



How does the model know what to do?

- So far: what to do is apparent from the input (e.g., which game is being played)
- What if the policy can do *multiple* things in the *same* environment?

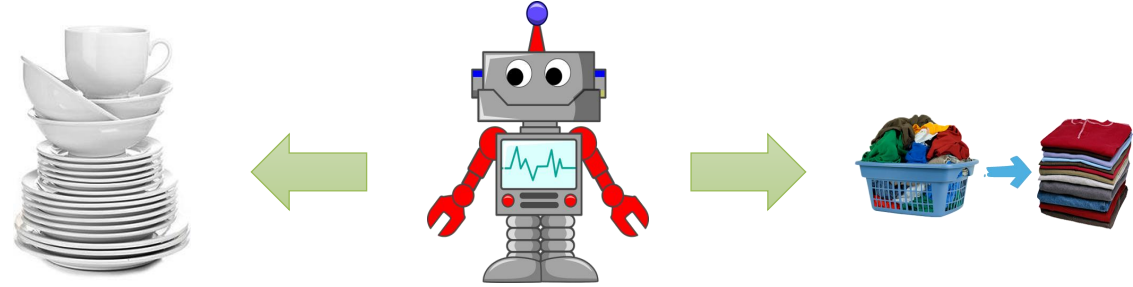


Contextual policies

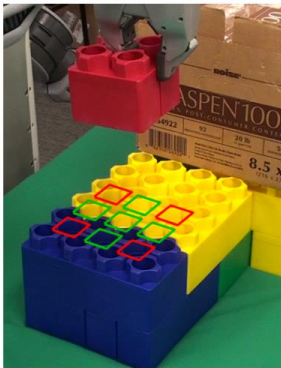
standard policy: $\pi_{\theta}(\mathbf{a}|\mathbf{s})$

contextual policy: $\pi_{\theta}(\mathbf{a}|\mathbf{s}, \omega)$

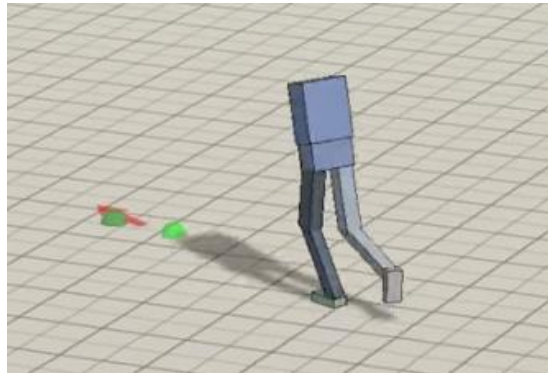
e.g., do dishes or laundry



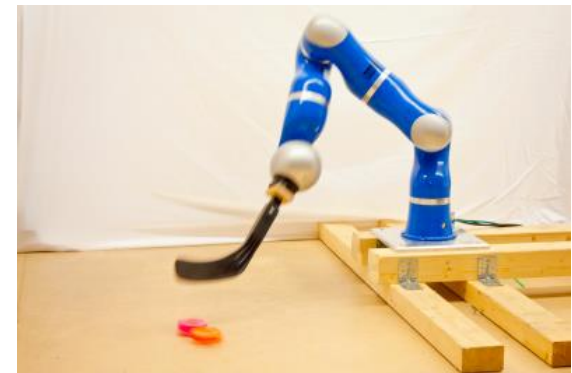
formally, simply defines augmented state space: $\tilde{\mathbf{s}} = \begin{bmatrix} \mathbf{s} \\ \omega \end{bmatrix}$ $\tilde{\mathcal{S}} = \mathcal{S} \times \Omega$



ω : stack location



ω : walking direction

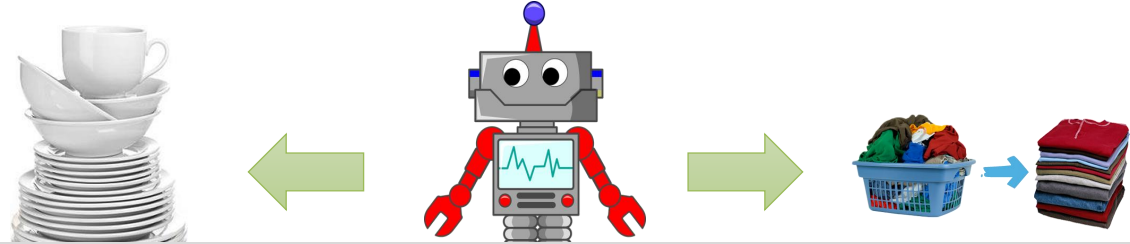


ω : where to hit puck

Contextual policies

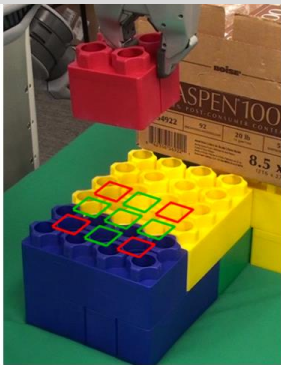
standard policy: $\pi_{\theta}(\mathbf{a}|\mathbf{s})$

contextual policy: $\pi_{\theta}(\mathbf{a}|\mathbf{s}, \omega)$

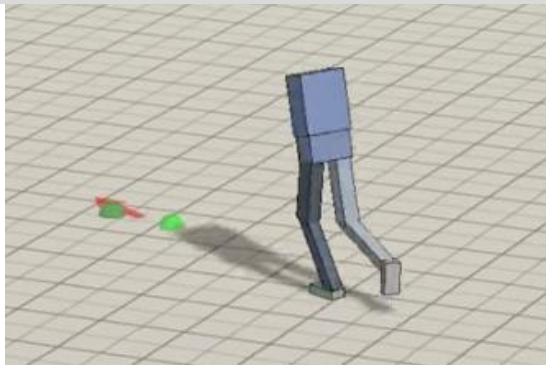


will discuss more in the context
of meta-learning!

for



ω : stack location



ω : walking direction



ω : where to hit puck

Transferring Models and Value Functions

The problem setting

Assumption: the **dynamics** $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ is the same in both domains
but the **reward function** is different

Common setting:

- **Autonomous car** learns how to drive to a few destinations, and then has to navigate to a new one
- A kitchen robot learns to cook many different recipes, and then **has to cook a new one** in the same kitchen

What is the best object to transfer?

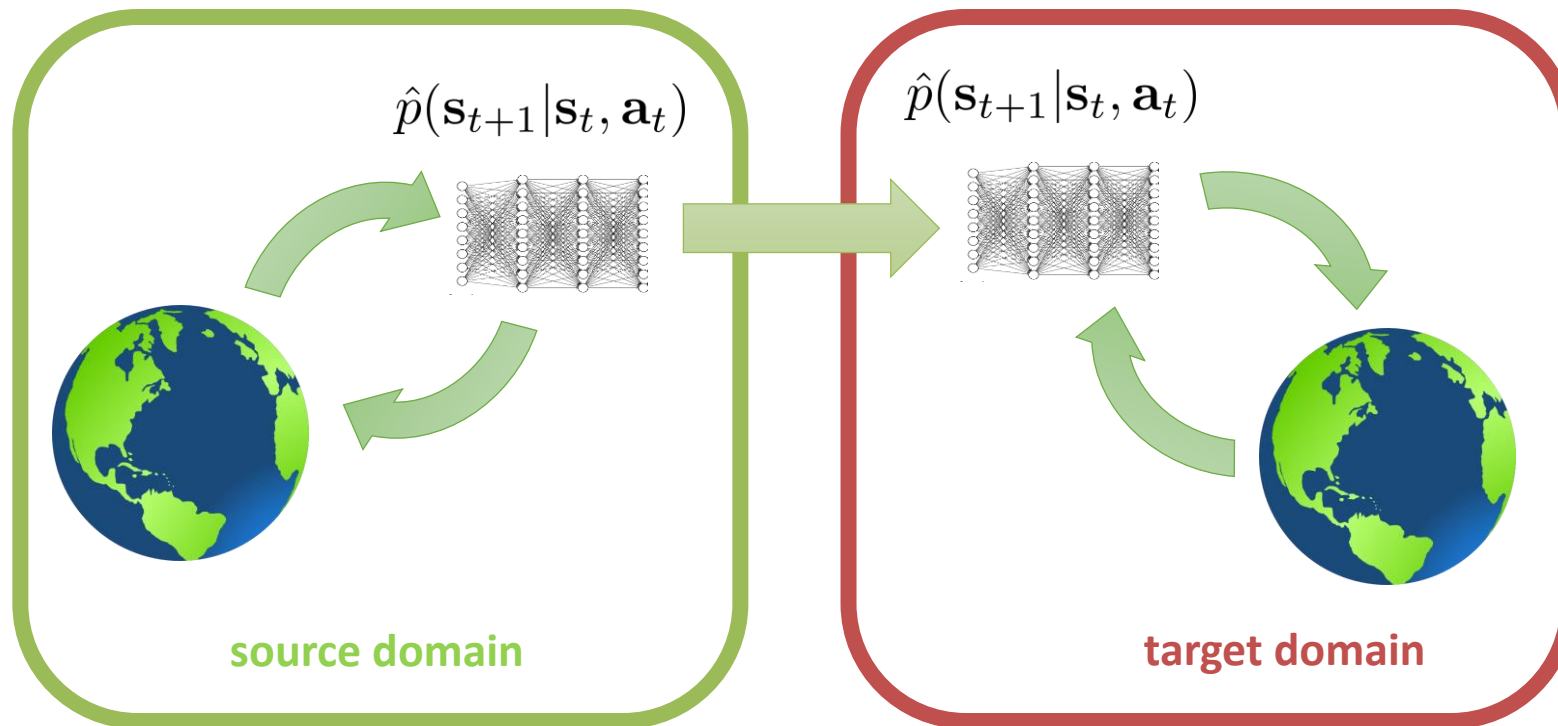
Model: very simple to transfer, since the model is already (in principle) independent of the reward

Value function: not straightforward to transfer by itself, since the value function entangles the dynamics and reward, but possible with a decomposition

- what kind of “dynamics relevant” information does a value function contain?

Policy: possible to do with contextual policies, but otherwise tricky, because the policy contains the *least* dynamics information

Transferring models



why might zero-shot transfer
not always work?

Transferring value functions

Not so fast! Value functions couple **dynamics**, **rewards**, and **policies**!

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \underbrace{r(\mathbf{s}, \mathbf{a})}_{\text{rewards}} + \gamma \underbrace{E_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})}}_{\text{dynamics}} \underbrace{E_{\mathbf{a}' \sim \pi(\mathbf{a}'|\mathbf{s}')}}_{\text{policies}} [Q^\pi(\mathbf{s}', \mathbf{a}')]]$$


Is this really such a good idea? **Yes, because of linearity**

Key observation: the value function is linear in the reward function

let $\mathbf{P}\mathbf{v}$ denote a vector \mathbf{w} of length $|S||A|$ given by $\mathbf{w}(\mathbf{s}, \mathbf{a}) = E_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})}[\mathbf{v}(\mathbf{s}')]]$

let $\mathbf{P}^\pi \mathbf{v}$ denote a vector \mathbf{w} of length $|S||A|$ given by $\mathbf{w}(\mathbf{s}, \mathbf{a}) = E_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a}), \mathbf{a}' \sim \pi(\mathbf{a}'|\mathbf{s}')} [\mathbf{v}(\mathbf{s}', \mathbf{a}')]]$

$$Q^\pi = r + \gamma \mathbf{P}^\pi Q^\pi \qquad Q^\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1} r$$


vectors with $|S||A|$ entries

Successor representations & successor features

$$Q^\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1} r$$

let ϕ be a $|S||A| \times N$ *feature* matrix

let ψ be a $|S||A| \times N$ matrix such that $\psi = (\mathbf{I} - \mathbf{P}^\pi)^{-1} \phi$

if $r = \phi w$, then $Q^\pi = \psi w$

$1 \times N$ row vector

Proof: $Q^\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1} r$

$$Q^\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1} \phi w$$

$$Q^\pi = \psi w$$

ψ_i is a “successor feature” for ϕ_i

Successor representations & successor features

let ϕ be a $|S||A| \times N$ *feature* matrix

let ψ be a $|S||A| \times N$ matrix such that $\psi = (\mathbf{I} - \mathbf{P}^\pi)^{-1}\phi$

if $r = \phi w$, then $Q^\pi = \psi w$

For any *new* reward function, if we can fit $r \approx \phi w$, we get $Q^\pi \approx \psi w$

Important: this holds for Q^π , *not* Q^* ! why?

$$Q^*(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})} [\max_{\mathbf{a}'} Q^\pi(\mathbf{s}', \mathbf{a}')]]$$

this is no longer linear!



Aside: successor representations

let ϕ be a $|S||A| \times N$ *feature* matrix

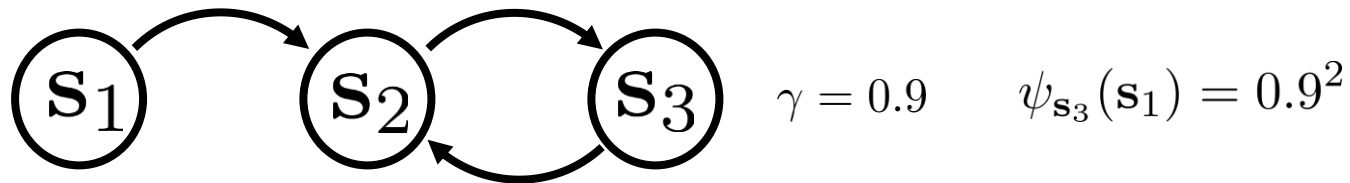
let ψ be a $|S||A| \times N$ matrix such that $\psi = (\mathbf{I} - \mathbf{P}^\pi)^{-1}\phi$

if $r = \phi w$, then $Q^\pi = \psi w$

what if $\phi = \mathbf{I}$? for each (\mathbf{s}, \mathbf{a}) , there is a $\phi_{\mathbf{s}, \mathbf{a}} = \delta(\mathbf{s}, \mathbf{a})$

then we can show that $\psi_{\mathbf{s}', \mathbf{a}'}(\mathbf{s}, \mathbf{a})$ predicts

probability of landing in $(\mathbf{s}', \mathbf{a}')$ from (\mathbf{s}, \mathbf{a}) under discount γ



Transfer with successor features

Simplest use: evaluation

1. get small amount of data $(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)$ in new MDP
2. fit w such that $\phi(\mathbf{s}_i, \mathbf{a}_i)w \approx r_i$ (linear regression)
3. initialize $Q^\pi(\mathbf{s}, \mathbf{a}) = \psi(\mathbf{s}, \mathbf{a})w$
4. finetune π and Q^π with any RL method

More sophisticated use: train multiple ψ^{π_i} functions for different π_i

choose initial policy $\pi(\mathbf{s}) = \arg \max_{\mathbf{a}} \max_i \psi^{\pi_i}(\mathbf{s}, \mathbf{a})w$

this provides a *better* initial policy in general

Recap

No single solution! Survey of various recent research papers

1. Forward transfer: train on one task, transfer to a new task
 - a) Transferring visual representations & domain adaptation
 - b) Domain adaptation in reinforcement learning
 - c) Randomization
2. Multi-task transfer: train on many tasks, transfer to a new task
 - a) Sharing representations and layers across tasks in multi-task learning
 - b) Contextual policies
 - c) Optimization challenges for multi-task learning
 - d) Algorithms
3. Transferring models and value functions
 - a) Model-based RL as a mechanism for transfer
 - b) Successor features & representations